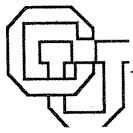
Branch Prediction using Selective Branch Inversion

Srilatha Manne
Artur Klauser
Dirk Grunwald

CU-CS-882-99



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAR 1999		**00-03-1999 to 00-03-1999**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Branch Prediction using Selective Branch Inversion	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Colorado at Boulder,Department of Computer Science,Boulder,CO,80309-0430**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**28**	

Branch Prediction using Selective Branch Inversion

Srilatha Manne

Department of Electrical
and Computer Engineering
Campus Box 425
University of Colorado
Boulder, CO 80309-0425
Email:bobbie@cs.colorado.edu

Artur Klauser and Dirk Grunwald

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309-0430
Email:{klauser,grunwald}@cs.colorado.edu

CU-CS-882-99

March 1999

University of Colorado at Boulder

Technical Report CU-CS-882-99
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

Branch Prediction using Selective Branch Inversion

Srilatha Manne

Department of Electrical
and Computer Engineering
Campus Box 425
University of Colorado
Boulder, CO 80309-0425
Email:bobbie@cs.colorado.edu

Artur Klauser and Dirk Grunwald

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309-0430
Email:{klauser,grunwald}@cs.colorado.edu

March 1999

Abstract

In this paper, we describe a family of branch predictors that use confidence estimation to improve the performance of an underlying branch predictor. With this method, referred to as *Selective Branch Inversion* (SBI), a confidence estimator determines when the branch predictor is likely to be incorrect; branch decisions for these low-confidence branches are inverted.

We show that SBI with an underlying Gshare branch predictor and an optimized confidence estimator outperforms other equal sized predictors such as the best Gshare predictor and Gshare with dynamic history length fitting, as well as equally complex McFarling, Bi-Mode, and Gskewed predictors. Our analysis shows that SBI achieves its performance through conflict detection and correction, rather than through conflict avoidance as some of the previously proposed predictors such as Bi-Mode and Agree. We also show that SBI can be used with other underlying branch predictors, such as McFarling, to improve their performance even further. Finally we show that *Dynamic Inversion Monitoring* (DIM) can be used as a safeguard to turn off SBI in cases where it degrades t he overall performance when compared to the underlying predictor.

1 Introduction

Branch prediction is an essential tool for current microprocessors. Even a reasonable branch misprediction rate can detrimentally hinder performance in a speculative, multi-issue processor. In [4, 2], the authors propose and evaluate a technique called *confidence estimation* for assessing the quality of a branch prediction. They suggest that confidence estimation can be used for speculation control in areas such as energy reduction [9] or controlling multipath execution [7]. In [4], the authors

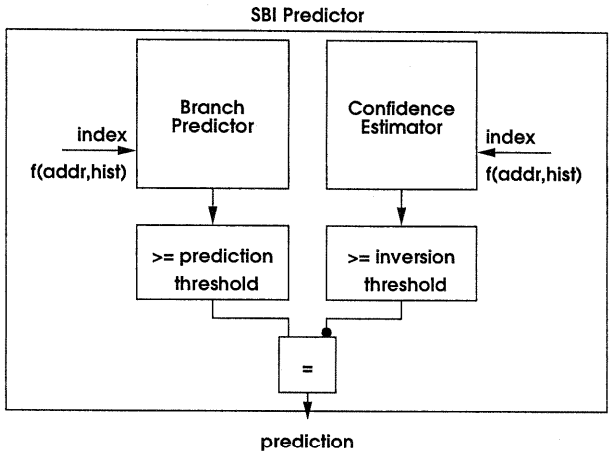


Figure 1: Schematic diagram showing how Selective Branch Inversion works.

propose that confidence estimation could also be used to improve the performance of branch predictors by inverting the branch prediction when the confidence estimator assesses the branch to be low-confidence. We call this mechanism *Selective Branch Inversion* (SBI).

Figure 1 shows how the SBI predictor works. We access a conventional branch predictor, such as a Gshare, McFarling or Bi-Mode predictor, in parallel with a confidence estimator. The branch prediction counter is compared to the prediction threshold to predict taken or not-taken. The confidence counter is compared to the inversion threshold. If the confidence counter is greater than or equal to the threshold, the branch prediction is labeled "high-confidence", meaning that the branch prediction is likely to be correct. Otherwise, the prediction is labeled "low-confidence" and the prediction is believed to be incorrect. The branch prediction is inverted for all low-confidence branches.

In [2], the authors analyze a series of confidence estimators and conclude that the prediction rate would worsen if confidence estimation were used for SBI because the evaluated confidence estimators could not identify mispredicted branches with sufficient accuracy. In this paper, we show that SBI can improve the branch prediction rate of several underlying branch predictors. The structure of the confidence estimator we use is similar to the mechanism proposed in [4], but modifications were necessary to make the confidence estimator work well for SBI.

The contributions of this paper are:

- We show that SBI improves many branch predictors, such as *Gshare*, *McFarling*, and *Bi-Mode*.
- We show that SBI is more efficient than other combination predictors. In particular, we show that combining a confidence estimator with a Gshare component out-performs a McFarling predictor, a Bi-Mode predictor, and a Gskewed predictor for the same hardware budget.
- We show that SBI is hardware efficient – adding a confidence estimator is more efficient than simply increasing the size of the underlying branch predictor.
- We analyze the source of mispredictions in Gshare and SBI and show that the majority of mispredictions are based in destructive interference in the branch predictor. SBI improves performance by detecting and correcting many of these conflict cases.

- We describe and use a method to dynamically monitor the confidence estimator to reduce the number of times when the confidence estimator makes an incorrect branch inversion.
- We justify our results using functional and pipeline-level simulations. We use fast functional simulation to explore a large parameter search space, and then use pipeline-level simulations to demonstrate the performance of the resulting branch predictors in a speculative, out-of-order microprocessor.

The paper is organized as follows. In Section 2, we describe related work and briefly introduce confidence estimators. In Section 3, we discuss the experimental mechanisms used in our work. Results and an analysis of the SBI predictor are presented in Section 4, and Section 5 concludes our work.

2 Background and Prior Work

2.1 Branch Predictors

We use four underlying branch predictors to compare the effectiveness of confidence estimators: a Gshare predictor [10], a Bi-Mode predictor [8], a McFarling predictor [10, 6] and a Gskewed predictor [11]. Figure 2 gives a schematic illustration of each branch predictor.

The Gshare branch predictor (Figure 2a) combines a global history with the program counter to select a two-bit counter from the pattern history table (PHT). A number of indexing schemes can be used, combining different parts of the program counter with the branch history register (BHR), and these have varying effects on the branch predictor performance [12]. We use the indexing method originally described in [10], which aligns the BHR to the most significant bits of the index and XORs it with the PC. The PC is stripped of its least significant zero-bits. We vary the number of history bits used for different experiments and we indicate how many history bits are used when we present results. This indexing method resulted in the best overall performance.

The Gskewed predictor (Figure 2b) uses three different skewing functions [11] to combine the program counter and global history to index into three different PHTs. We implement majority update for all simulations. We also tried the enhanced Gskewed predictor, which replaces one component predictor with a bimodal predictor[1], but it resulted in a decreased average performance for our benchmarks. Thus, we do not show any results for this configuration.

The McFarling predictor (Figure 2c) combines a Gshare (Figure 2a) and a bimodal predictor. It uses a "meta-predictor" consisting of two-bit up-down counters that selects which component branch predictor to use for each prediction. Both component predictors are updated for each branch. Similarly, the Bi-Mode predictor (Figure 2d) uses two Gshare predictors and a meta-predictor to select between the two component predictors. The meta-predictor is indexed by the program counter. The Bi-Mode predictor only updates the Gshare component that is used for a branch prediction. The choice predictor is updated with the branch outcome, except when the choice is opposite to the branch outcome, but the selected counter of the direction predictor makes a correct final prediction [12].

The global history of all predictors is updated speculatively, i.e. it is updated with the predicted branch outcome, and later restored if the branch is mispredicted. Non-speculative update increases

[1] A *bimodal* predictor is simply a pattern history table indexed by the program counter.

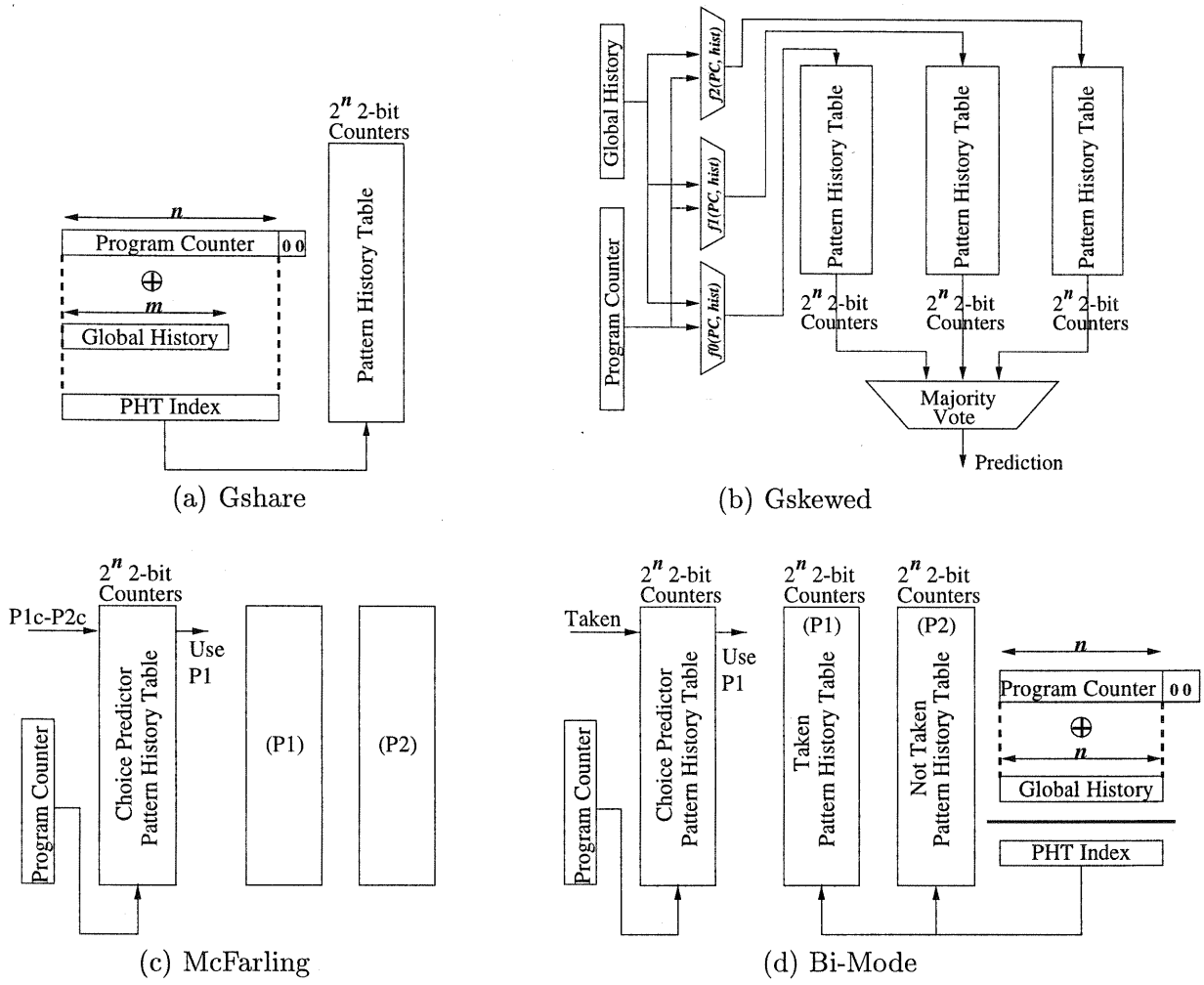


Figure 2: Schematic illustration of the different branch predictors.

the branch misprediction rate, because information from recent branches is not immediately available to succeeding branches [3].

2.2 Confidence Estimators

Jacobsen, Rotenberg, and Smith [4] proposed a confidence estimator that uses a miss distance counter (MDC) table. Each MDC keeps track of the number of correct predictions since the last misprediction for this counter. We call this the JRS confidence estimator. The structure of the confidence estimator is similar to that of the Gshare predictor. An index is computed using an exclusive-or of the program address and the branch history register, similar to the index computation used in the Gshare branch predictor. However, the history width can vary between the Gshare predictor and the JRS confidence estimator. This index is used to read a value from a table of MDCs. The width of these counters can vary in size, and four bit counters with varying thresholds were used in [4]. Each time a branch is predicted, the MDC value is compared to the inversion threshold. If the value is greater than or equal to the threshold, then the branch is labeled high-

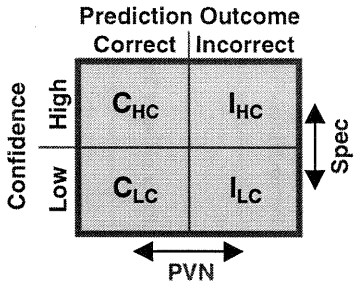


Figure 3: Confidence estimator quadrant table.

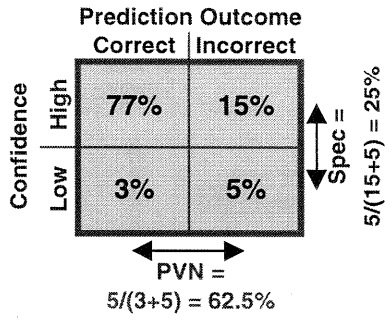


Figure 4: Example confidence estimator performance.

confidence, otherwise it is labeled low-confidence. When a branch resolves, the corresponding MDC is incremented if the branch was correct; otherwise, it is reset to zero. A number of parameters, such as the width of the MDC and the inversion threshold at which branches are considered high-confidence, can be adjusted.

We made a number of modifications to this design to improve the effectiveness of the confidence estimator for SBI. Before describing those changes in Section 4.1, we need to understand how prior work has evaluated confidence estimators. Following the notation of [2], we note that branch predictors either make *correct* or *incorrect* predictions. For each prediction, a confidence estimator indicates low-confidence or high-confidence. All branches fall into one of four categories in this space. Again, using the notation of [2], we draw a 2×2 matrix listing the frequency for each outcome of a test and normalize the values to insure that the sum equals one (see Figures 3 and 4). In the figures, "C" and "I" refer to correct and incorrect predictions, respectively, and "HC" refers to high-confidence and "LC" to low-confidence. During a simulation, we measure C_{HC}, I_{HC}, C_{LC} and I_{LC} using a branch predictor and a confidence estimator. When the branch is resolved, we classify the branch as belonging to class C_{HC}, I_{HC}, C_{LC} or I_{LC}.

SBI inverts the branch prediction when the confidence estimator indicates a low-confidence branch. All low-confidence branches that were predicted incorrectly (I_{LC}) will be predicted correctly after the inversion, and all low-confidence branches that were predicted correctly (C_{LC}) will be predicted incorrectly after the inversion. Through SBI, I_{LC} and C_{LC} change place in Figure 3, resulting in $(I_{LC} - C_{LC})$ more branches being predicted correctly. We define the expression $(I_{LC} - C_{LC})/(I_{LC} + I_{HC})$ as the INVERSIONBENEFIT, which expresses the reduction in branch misprediction rate when inversion is applied. SBI is only effective if I_{LC} is greater than C_{LC}, i.e. the INVERSIONBENEFIT is positive. Otherwise SBI results in a performance loss. For example, in Figure 4 the misprediction rate before inversion is $15\% + 5\% = 20\%$. With the INVERSIONBENEFIT of 10% (*i.e.* $(5\% - 3\%)/20\%$), we get a misprediction rate after inversion of 18% (*i.e.* $20\% - 2\%$).

To understand the performance implications of various parameters in the confidence estimator design-space, we also use two metrics introduced in [2]. The *predictive value of a negative test*, or the PVN, is the probability that a branch is incorrectly predicted if it was low-confidence. Following the notation of [2], PVN can be written as the conditional probability $P[I|LC] = I_{LC}/(C_{LC} + I_{LC})$. For the interesting range of positive INVERSIONBENEFIT, we get a PVN $> 50\%$. However, the PVN

only indicates the probability of correctly classifying mispredicted branches – it does not indicate how many mispredicted branches are found by the confidence estimator. For example, assume a program executes 100 branches, and only one branch is marked as low-confidence. Further assume that this branch is mispredicted along with 9 other branches. The PVN is 100%, but the overall speedup by applying SBI is small because only one of 10 mispredicted branches is inverted. Thus, another important metric in determining the quality of SBI is the *specificity* (SPEC). SPEC is $P[LC|I] = I_{LC}/(I_{HC} + I_{LC})$, or the fraction of incorrect predictions identified as low-confidence. In our previous example, the SPEC is 10%. Ideally we want both PVN and SPEC to be as high as possible.

3 Experimental Methodology

We used the SimpleScalar [1] execution-driven simulation infrastructure to study the applicability of SBI. SimpleScalar was modified to implement a wide variety of branch predictors and confidence estimators. We used a functional branch simulator (a modified version of *sim-bpred*) to measure the non-speculative execution of the processor. This simulator is significantly faster than the full pipeline model and allows us to explore a wide parameter space.

The pipeline-level simulator is an extension of the *sim-outorder* simulator distributed with the SimpleScalar tools. Table 1 shows the processor configuration, which is designed as a current-generation, speculative, 4-way superscalar, out-of-order RISC CPU. The minimum branch misprediction penalty is 6 cycles. The actual misprediction latency depends on the number of cycles a branch waits in the instruction window for data dependencies to resolve.

We used the SPECint95 benchmarks for our performance evaluation. We did not simulate the SPECfp95 benchmarks because those programs typically pose few difficulties for branch predictors. The benchmarks and some characteristics are listed in Table 2. Since we need consistent inputs between the fast functional simulator and the slower, pipeline-level simulator, we use reduced inputs to run each program to completion. The inputs are either the train input set or versions that further reduce the execution time. All benchmarks are compiled with full optimization.

4 Results

In this section, we first present a confidence estimator that consistently achieves a PVN greater than 50% across a number of applications. Section 4.1 describes this confidence estimator. Subsequently we compare the efficacy of SBI with a Gshare branch predictor, and analyze why the SBI predictor works. We then extend this work to other underlying branch predictors.

4.1 Selecting a Confidence Estimator

When designing a confidence estimator for SBI, it is important to insure that the PVN is greater than 50% and to increase the PVN and SPEC as much as possible, i.e. to maximize the INVERSIONBENEFIT. If the PVN falls below 50%, then the value of the SPEC is unimportant; in this case SBI should not be used because it decreases performance. We use the INVERSIONBENEFIT metric to assess the misprediction improvement achieved by SBI. We refer to the PVN and SPEC metrics because they provide more insight into *why* changes to the confidence estimator influence SBI performance.

Pipeline Simulator Configuration	
Machine Width	4-wide fetch, 4-wide issue, 4-wide commit
Window Size	64 entry register update unit, 32 entry load/store queue
Branch Misprediction	min. recovery latency 6 cycles
L1 Icache	64 kB, 32 byte lines, 2-way set-associative, 2 cycles hit latency
L1 Dcache	64 kB, 32 byte lines, 2-way set-associative, 2 cycles hit latency
L2 Cache Combined	512 kB, 64 byte lines, direct mapped, 6 cycles hit latency
Memory	128 bit wide, 26 cycles access latency
BTB	1024 entry, 4-way set-associative; 32 entry return address stack
TLB	64 entry (I), 128 entry (D), fully associative, 30 cycle miss latency
Functional Units and Latency (total/issue)	4 Int.ALU (1/1), 1 Int Mult (3/1) / Div (20/19), 2 Ld./St. (2/1), 4 FP Add (2/1), 1 FP Mult (4/1) / Div (12/12) / Sqrt (24/24)

Table 1: Configuration parameters of the detailed pipeline-level simulation.

	Input Data Set	Instr. (Millions)	Branches (Millions) all	cond.
compress	train **	80.4	14.4	9.0
gcc	varasm	250.9	50.4	38.2
perl	jumble*	262.3	50.2	35.2
go	2stone9	548.2	80.3	62.1
m88ksim	dhry	416.5	89.8	68.7
lisp	train	183.3	41.8	24.2
vortex	train*	180.9	29.1	21.1
jpeg	specmun*	252.0	20.0	16.4

Benchmarks (*) have reduced input data sets
Benchmark (**) has increased input data set

Table 2: Program characteristics of the benchmark applications.

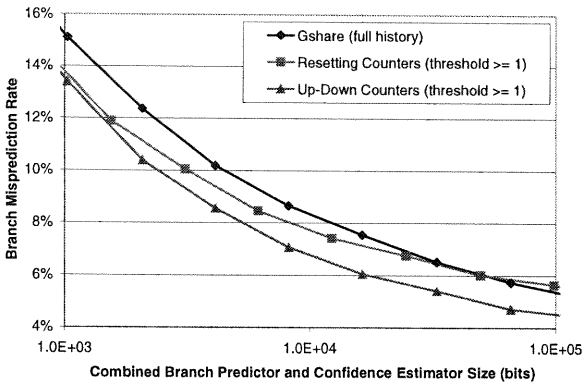


Figure 5: Geometric means of the branch misprediction rates of the original JRS confidence estimator and the modified version used for SBI.

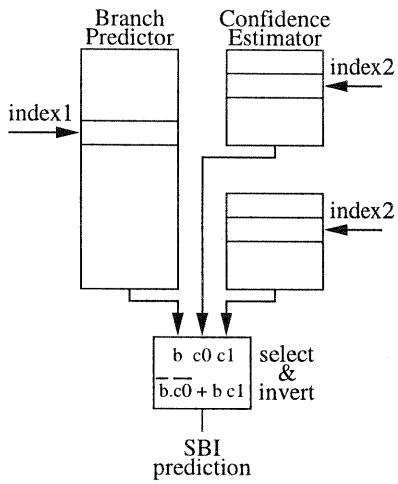


Figure 6: Schematic illustration showing how the confidence estimator is integrated with the branch predictor

Figure 5 shows the branch misprediction rate for the Gshare branch predictor using the full branch history *vs.* a Gshare predictor using SBI with two different confidence estimators. The first estimator, labeled "Resetting", is similar to the JRS mechanism proposed in [4] and uses a one-bit resetting counter. Klauser et al. [7] also used a single-bit resetting JRS mechanism to control multipath execution, because that configuration results in the highest PVN for resetting counters. As described in [2], the outcome of the branch prediction made by the Gshare predictor is used as part of the branch history for the JRS indexing function. We accomplish this by splitting up the confidence estimator and reading both halves of the table (c_0, c_1) concurrently with the branch predictor (b), as shown in Figure 6. If the MDC is one, the branch has high-confidence. The three predictions are combined using a three-input, one-output logic block labeled "select & invert" ($\overline{b}\overline{c}_0 + bc_1$). Similar select logic is used by the McFarling and Bi-mod predictors, and the logic from the confidence estimators can be combined with the logic from multi-level predictors without an increase in gate delays.

The one-bit MDC results in the lowest branch misprediction rate when using resetting counters. The resetting counter exploits the temporal distribution of branch mispredictions in a program. As discussed in [2], branch mispredictions are clustered, and if one branch is mispredicted, future branches close to the mispredicted branch have a higher probability of being mispredicted. A low threshold (i.e., one in this example) reduces the number of low-confidence branches, increasing PVN and lowering SPEC.

The second estimator, labeled "Up-Down", uses two-bit up-down counters rather than resetting counters for the confidence estimator. The counters are indexed as before. The configuration shown has the best performance across a range of thresholds, but we did not examine larger counters. The up-down counters produce better misprediction rates for the same reason that branch predictors benefit from two-bit pattern history tables. Most branch predictors mispredict loop-exit branches; a two-bit PHT reduces the number of mispredictions from two to one for loop exit branches. Likewise,

	Spec				PVN				Inversion Benefit			
counter type	resetting		up/down		resetting		up/down		resetting		up/down	
threshold	>=2	>=1	>=2	>=1	>=2	>=1	>=2	>=1	>=2	>=1	>=2	>=1
compress	64.4%	44.5%	41.7%	27.8%	41.3%	44.5%	57.3%	66.6%	-27.1%	-11.1%	10.7%	13.9%
gcc	65.6%	52.6%	45.2%	32.0%	44.5%	52.6%	59.5%	70.6%	-16.2%	5.1%	14.4%	18.6%
perl	73.6%	51.3%	50.9%	43.3%	49.4%	51.2%	75.1%	84.7%	-1.9%	2.4%	34.1%	35.5%
go	66.4%	48.6%	41.3%	23.6%	43.9%	48.6%	50.9%	57.1%	-18.6%	-2.8%	1.4%	5.8%
m88ksim	57.4%	46.1%	41.3%	33.7%	37.2%	46.0%	67.9%	81.4%	-39.4%	-7.9%	21.8%	26.0%
lisp	57.4%	39.7%	34.8%	22.9%	35.7%	39.6%	54.7%	65.5%	-45.8%	-20.8%	6.0%	10.9%
vortex	78.6%	69.6%	67.7%	60.5%	60.0%	69.4%	80.9%	89.0%	26.2%	38.9%	51.7%	53.1%
jpeg	62.7%	44.8%	39.9%	23.9%	40.3%	44.8%	52.4%	59.6%	-30.1%	-10.5%	3.7%	7.7%

Table 3: SPEC, PVN, and INVERSIONBENEFIT for resetting and up-down counters for each application.

	gshare 8k			SBI
	full history	best avg. history (10)	best history	Predictor 4k+4k
compress	9.37%	9.56%	9.08% (12)	8.87%
gcc	16.43%	13.85%	10.90% (5)	11.06%
perl	3.46%	3.04%	3.04% (10)	2.57%
go	28.14%	24.76%	20.94% (5)	22.00%
m88ksim	4.51%	4.12%	4.10% (9)	3.73%
lisp	6.28%	6.51%	6.28% (13)	6.10%
vortex	3.08%	2.37%	1.61% (4)	1.33%
jpeg	11.32%	10.50%	10.07% (9)	10.26%

Table 4: Comparison of equivalent sized Gshare and SBI predictors. Numbers in parentheses indicate the number of history bits used in the index computation. These values were determined from the pipeline-level simulator.

exiting a loop is a poor indication that future branches will be mispredicted, making the up-down counters better than the resetting counters. In other words, exiting a single loop is not sufficient to move a saturated up-down counter below the threshold.

Table 3 shows the SPEC, PVN and INVERSIONBENEFIT values for the resetting and up-down counters with different thresholds. The values for PVN and SPEC shown in Table 3 for resetting counters are similar to those shown in [2]. This data reinforces the previous explanation; increasing the threshold or resetting on an incorrect prediction means that more of the mispredicted branches are low-confidence, but a larger fraction of the low-confidence branches are correctly predicted. The up-down counters consistently have a lower SPEC, but a significantly higher PVN, resulting in a higher INVERSIONBENEFIT.

4.2 Using SBI with the Gshare branch predictor

As a first step, we apply SBI to a Gshare branch predictor. Table 4 shows the misprediction rates for all benchmarks using variants of Gshare and Gshare combined with SBI. The Gshare predictor in Table 4 uses an 8k-entry PHT. As noted in [12], the performance of the Gshare predictor varies considerably with different history lengths. We simulated all possible history lengths, from 0 to 13 bits. The "full" column uses the largest possible history length (13 bits) to index the PHT.

	gshare 16k			Combination Predictors				
	full history	best avg. history (11)	best history	SBI 8k+8k	SBI 8k+4k	McFarling 4k+4k+4k	Bi-Mode 4k+4k+4k	Gskewed 4k+4k+4k
compress	8.81%	9.33%	8.62% (12)	8.47%	8.51%	9.40%	8.69%	10.01%
gcc	14.83%	12.51%	9.85% (6)	9.52%	10.30%	10.17%	10.02%	12.55%
perl	2.85%	2.42%	2.42% (11)	2.41%	2.27%	2.93%	2.51%	4.62%
go	25.24%	22.19%	19.33% (6)	19.87%	21.02%	22.24%	22.42%	22.93%
m88ksim	4.07%	3.84%	3.67% (9)	3.36%	3.47%	3.75%	3.43%	5.92%
lisp	5.78%	6.10%	5.78% (14)	5.81%	5.87%	6.33%	6.12%	7.89%
vortex	2.23%	1.60%	1.28% (3)	0.94%	1.13%	1.27%	1.06%	1.96%
jpeg	10.58%	10.07%	9.79% (10)	9.85%	10.09%	10.12%	10.20%	10.66%

Table 5: Comparison of different combination predictors. Numbers in parentheses indicate the number of history bits used in the index computation. The optimal configurations were determined from non-pipelined simulations, but the values shown are from the pipelined simulations.

The results in the "best avg." column use the single best history length that produces the lowest average misprediction rate across all applications. The results in the "best" column use the best history length *for each application.* The lengths are shown in parentheses.

The last column shows the misprediction rate when SBI is applied to a smaller (4k-entry) Gshare branch predictor. The total size of the SBI predictor and the larger Gshare predictors is the same, since both the branch predictor and the confidence estimator use 2-bit counters. The Gshare component of the SBI predictor is indexed using the full 12-bit history, while the confidence estimator uses three bits of branch history. In general, we found that the best indexing function for the Gshare predictor changes when using SBI. More importantly, indexing the confidence estimation with the same history length as the branch predictor consistently produces inferior performance.

These results show that using an SBI predictor with a single history size across all applications out-performs the Gshare predictor even when using the best history size *specific to each application.* Recently, Juan *et al*[5] proposed a mechanism to dynamically adjust the history length of the Gshare predictor to adapt it to the characteristics of individual programs. Their technique produces results typically close to, but usually worse than, the misprediction rate when using the best individual history length. Table 4 shows that SBI is generally better than Gshare with the best individual history; thus we would expect SBI to out-perform the more complex technique described in [5].

Table 5 compares the performance of Gshare to the McFarling, Bi-Mode, Gskewed and SBI predictors. Here, the Gshare predictor is configured with a 16k-entry PHT. Since the McFarling, Bi-Mode and Gskewed predictors use three tables, it is difficult to compare equally sized configurations. Thus, we use 12k-entries for each combination predictor, 4k-entries fewer than the Gshare predictor. Table 5 presents results for two configurations of SBI. The first uses an 8k-entry Gshare predictor and an 8k-entry confidence estimator, resulting in a SBI predictor that is comparable to the 16k-entry Gshare predictor. The second SBI configuration uses an 8k-entry Gshare and a 4k-entry confidence estimator. This predictor is comparable in size to the combination predictors in Table 5. The smaller SBI predictor has a better misprediction rate than the Bi-Mode predictor on 5 of 8 benchmarks; it always out-performs McFarling and Gskewed.

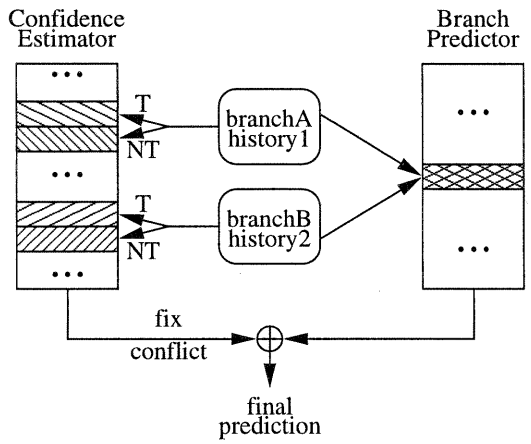


Figure 7: Branch predictor interference.

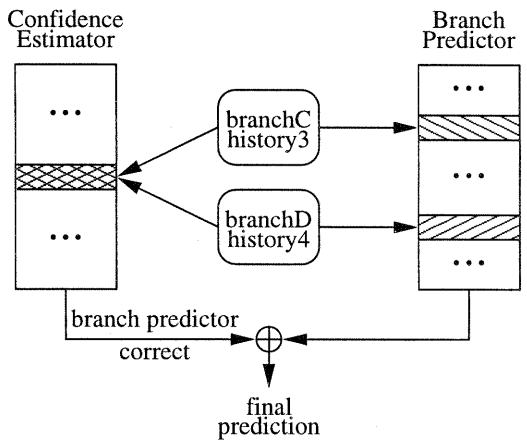


Figure 8: Confidence estimator interference.

4.3 Why SBI works

In this section we will present the reasoning behind SBI, why it works, and what the design decisions are that allow it to outperform other branch predictors. First, we will give an intuitive overview of the design of SBI, which is followed by a quantitative analysis of SBI and Gshare.

4.3.1 Qualitative Analysis

Interference occurs when the indices of two branches alias into the same location in the branch predictor. Interference in the branch predictor is a significant source of mispredictions [15]. Interference can be either constructive, neutral, or destructive. We call destructive interference cases *conflicts.*

SBI works by finding conflicts in the branch predictor and correcting the prediction outcome for these cases. Two branches which interfere in the branch predictor are unlikely to also interfere in the confidence estimator and vice versa; thus the different entries in the confidence estimator are trained to remember conflict situations in the branch predictor and alleviate the problem.

The SBI predictor has two sources of interference: branch predictor tables and confidence estimator tables. We first look at two branches and their histories, (branch A, hist 1) and (branch B, hist 2), that interfere in the branch predictor but not in the confidence estimator. Figure 7 depicts this situation. If the two branches are biased in the same direction, then the interference in the branch predictor is of little consequence to the prediction outcome (neutral interference) and can be ignored. However, if the two branches are biased in opposite directions, conflicts occur in the branch predictor, which results in an increased number of mispredictions for both branches. The two branches are unlikely to also interfere in the confidence estimator for several reasons:

- The branch predictor and confidence estimator use slightly different indexing schemes by shifting the history into different positions before it is XOR-ed with the address. A similar idea is used by the Gskewed predictor, which also uses different indexing functions for different predictor components [11]. Note, however, that the Gskewed predictor uses a large history

size for each indexing function, resulting in significant interference within each predictor component.

- The confidence estimator uses a much smaller history size than the branch predictor. This reduces the footprint of each branch in the confidence estimator, thus reducing the chances of interference.

Besides conflict detection, the confidence estimator needs to decide in which situations the outcome of the branch predictor should be inverted. Using the predicted outcome of the branch as another speculative bit of history for the indexing function results in four locations in the confidence estimator being used to track the behavior of the interfering location in the branch predictor (see Fig. 7). If the speculative bit were not used, only two locations in the confidence estimator would track the branch predictor conflict. The use of this "predicted" next bit of history effectively turns the N-bit history Gshare predictor into a N+1-bit history SBI predictor. A similar effect is used in the Bi-Mode predictor [8], which also uses a predicted bit of history from the *choice predictor* to index into the *direction predictor*.

The second source of interference in SBI is the confidence estimator. Figure 8 shows the situation of two branches, (branch C, hist 3) and (branch D, hist 4), interfering in the confidence estimator but not in the branch predictor. This situation can occur because the branch predictor has more history bits than the confidence estimator. A difference in any of the additional history bits results in the two branches occupying different entries in the branch predictor, despite their interference in the confidence estimator. In order to minimize the impact of interference in the confidence estimator, the SBI predictor uses the following properties:

- Small history: A small history in the confidence estimator indexing function results in a small footprint for each branch in the confidence estimator.

- Suppress spurious events: The confidence estimator uses up/down counters with a threshold of 1. It predicts high confidence for the counts of 1 to 3, and low confidence for the count of 0. By requiring three successive mispredictions to switch to the low confidence state, and due to the high accuracy of the branch predictor itself, fewer correctly predicted branches are marked low confidence.

- Fast recovery: A single correct prediction brings the confidence estimator back into high confidence state, which allows for fast recovery if the low confidence state was reached erroneously.

- "Agreeness": Since the confidence estimator stores "*agreeness*" with the branch predictor, rather than a direction, two oppositely biased branches aliasing in the confidence estimator result in neutral interference if they are predominantly predicted correctly. Only indeterminate branches result in conflicts in the confidence estimator entry. This is similar to concepts used in the Agree predictor [13], but does not require a static prediction.

4.3.2 Quantitative analysis

We have analyzed the behavior of a number of benchmarks on the SBI predictor and in this section we specifically present the analysis of *gcc* for demonstrative purposes. Other benchmarks show similar behavior.

To analyze the differences between different branch predictors and different configurations, we separate all branch predictions into two streams:

- **Interference Stream**: This stream contains all branches whose prediction is or could be influenced by aliasing in the branch predictor. A branch prediction belongs to the interference stream if any of two properties are true:
  - Direct interference: The branch is predicted by an index in the branch predictor that was last updated by a different branch. This interference can be constructive, neutral, or destructive.
  - Indirect interference: The prediction derived from the branch predictor differs from the prediction that would be derived from a predictor that has a unique prediction table assigned to each static branch. This category covers interference that has happened in the past and has significantly changed the state of the branch predictor. This interference can be either constructive or destructive.

  Note that the direct and indirect interference categories are overlapping, but neither is a subset of the other.

- **Non-Interference Stream**: This stream contains all other branch predictions that are not considered to be generated by interference.

In each substream we count the number of correct and incorrect predictions. We present all data in percent, normalized to the total number of predictions, i.e. the misprediction rates of the substreams add up to the total misprediction rate.

First we analyze an 8k-entry Gshare predictor with varying history size and consider what happens if we double the size to 16k-entries. Figure 9 shows these results. The graph shows the misprediction rate contribution of the interference stream (I-stream), the non-interference stream (NI-stream), and the total misprediction rate which is the sum of the former two. We see in the graph that the contribution from the NI-stream dominates for small history sizes, and gets smaller as the history increases. Many branches are not predictable with a small history, but can be predicted better with longer histories. On the other hand, the contribution from the I-stream is small for small history sizes and increases for larger histories. With small histories, each branch instruction has a small footprint in the branch predictor and thus a smaller chance of destructive interference. With large histories, however, the predictor suffers severely from conflicts, so the misprediction contribution from the I-stream is high. The overall misprediction rate, the sum of the NI-stream and the I-stream, has a minimum at a history size of 5 bits, where the tradeoff between better predictability and more conflicts due to larger histories reaches its optimum. This optimal history size differs for different benchmarks, depending on the number and prediction pattern of their branches.

If we now double the size of the branch predictor to 16k-entries, we see that the misprediction contribution of the NI-stream changes very little, whereas the misprediction contribution of the I-stream is dramatically reduced. The improvement of the I-stream comes from the reduction of interference in the predictor due to a larger predictor size. The small difference in the NI-stream can be explained by the difference in interference patterns between the 8k and 16k predictors. The NI-stream does not cover exactly the same branches in both cases. The 16k-entry predictor shows a

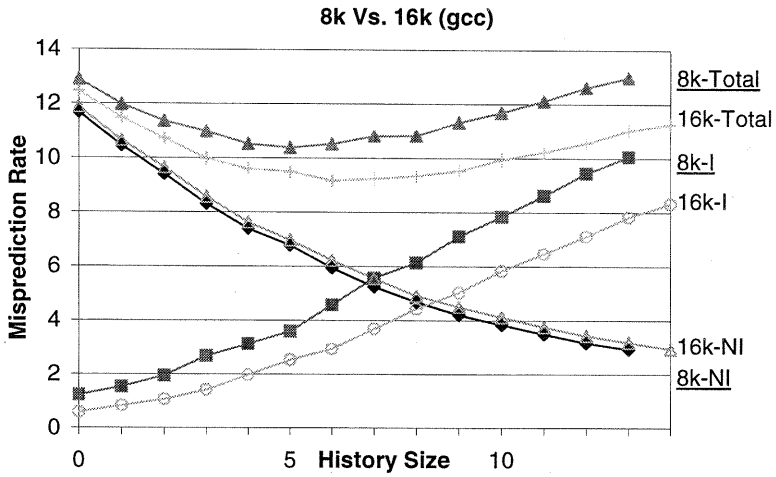


Figure 9: Gcc: 8k-entry Gshare compared to 16k-entry Gshare predictor.

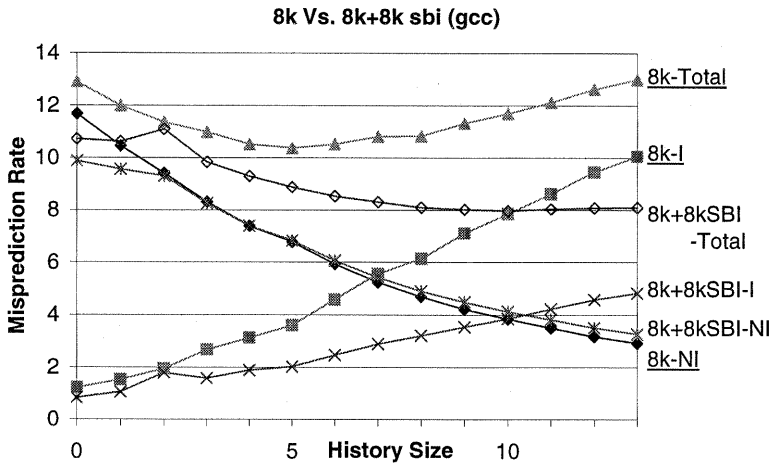


Figure 10: Gcc: 8k-entry Gshare compared to 8k+8k-entry SBI predictor.

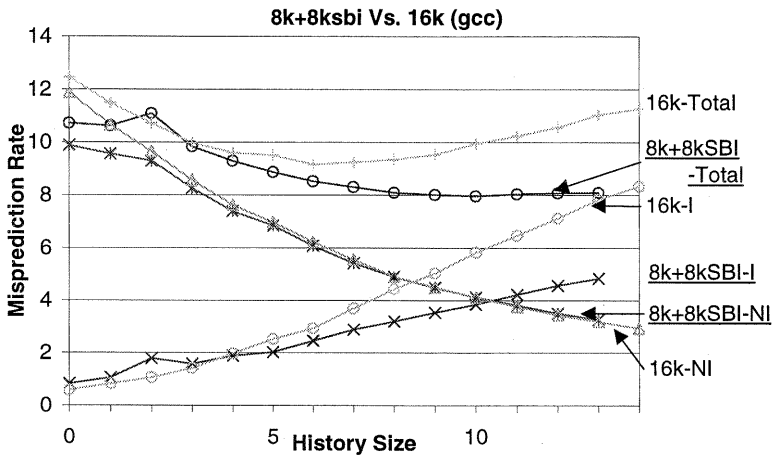


Figure 11: Gcc: 16k-entry Gshare compared to 8k+8k-entry SBI predictor.

reduced misprediction rate and a larger optimal history size, predominantly due to the interference reduction in the larger predictor.

Figure 10 shows the same analysis as before, but instead of doubling the size of the Gshare predictor, we add an 8k-entry confidence estimator on top of the 8k-entry Gshare component (8k+8kSBI). For the SBI predictor, the history size used to index into the confidence estimator is kept constant at 3 bits, whereas the history size used to index into the Gshare component varies as indicated on the X-axis. The interference and non-interference streams measure the interference in the branch predictor, not in the confidence estimator.

For large histories, we see the same trend as before. The SBI predictor is very effective in reducing I-stream mispredictions, while keeping the NI-stream mispredictions close to that of the smaller predictor. For smaller history sizes the interference reduction is much smaller, so the overall benefit is also smaller.

An interesting region emerges for history sizes ≤ 2. Since the confidence estimator is always indexed with 3 bits of history, for X-axis values ≤ 2, the confidence estimator has more history than the Gshare predictor. In this domain, the branch predictor has worse prediction accuracy than the confidence estimator alone, so the confidence estimator takes over the majority of the workload and the branch predictor acts only as a helper to produce one more speculative bit of history for the confidence estimator. This is similar to operation of the Bi-Mode predictor [8] which uses a history-less *choice predictor* (bimodal) used to generate an additional speculative history bit for the *direction predictor* indexing function.

At a history size of two, both the confidence estimator and the branch predictor align the same history bits over the same address bits in their indexing computation, with the confidence estimator having one additional speculative bit of history. In this situation SBI accrues almost no benefit over the 8k-entry Gshare. Using aligned histories severely limits the ability of the confidence estimator to detect conflicts in the branch predictor, since it suffers from much the same interferences. For history sizes greater than two, SBI operates in its preferred domain of detecting conflicts in the branch predictor and fixing their prediction outcome, as can be seen by the large reduction of the I-stream misprediction contribution. Also note that for large histories the NI-stream performs slightly worse due to a small number of incorrect decisions in the confidence estimator for the non-interference stream. This is likely due to interference in the confidence estimator, but we did not investigate it any further since it has a small impact on the misprediction rate.

Finally, Figure 11 compares the 16k-entry Gshare predictor with the 8k+8k-entry SBI predictor, which provides an equal bit-cost comparison. Here we see that the best overall misprediction rate for SBI is reached for a history of 11 bits; however the misprediction rate for full history (13-bits) is almost identical. Concentrating on large histories, we see that SBI is much more effective in reducing mispredictions in the I-stream than the same size Gshare predictor. On the other hand, the misprediction contribution from the NI-stream is virtually identical in both cases. The NI-stream mispredictions originate from branches that are inherently unpredictable with the particular history information, which is why they can not be reduced. The I-stream mispredictions are based on the way a branch predictor is built, and SBI has a clear performance advantage in this case.

This concludes our analysis that has shown that SBI works by detecting and correcting conflicts in the branch predictor, and it is more effective than just increasing the size of the branch predictor.

	Spec	PVN	Inversion Benefit
compress	24.2%	62.7%	0.9%
gcc	20.7%	64.3%	0.8%
perl	25.0%	75.6%	0.6%
go	17.6%	54.7%	0.7%
m88ksim	16.1%	67.3%	0.3%
lisp	17.0%	58.9%	0.3%
vortex	31.9%	75.5%	0.3%
jpeg	20.7%	54.5%	0.4%

Table 6: Performance of a 4k-entry SBI with an underlying McFarling predictor of size 4k+4k+4k entries.

	Spec	PVN	Inversion Benefit
compress	18.6%	55.7%	0.3%
gcc	18.5%	61.0%	0.6%
perl	17.8%	68.2%	0.3%
go	17.5%	53.8%	0.6%
m88ksim	15.5%	62.3%	0.2%
lisp	15.4%	56.0%	0.2%
vortex	25.8%	70.5%	0.2%
jpeg	20.8%	54.5%	0.4%

Table 7: Performance of a 4k-entry SBI with an underlying Bi-Mode predictor of size 4k+4k+4k entries.

McFarling Configuration	Misprediction Rate (Geo Mean)
Alpha 21264 (29k)	5.47%
21264 Modified (21k)	5.91%
21264 Modified + 4k-entry SBI (29k)	4.97%

Table 8: Comparison of different McFarling predictors. The misprediction rate shown is the geometric mean over all benchmarks. The "Alpha 21264" refers to the McFarling predictor used in the 21264 processor [6]. The "21264 Modified" predictor uses a smaller configuration of the 21264 predictor. The last row refers to the modified 21264 predictor with a 4k-entry confidence estimator. The bit size of each predictor is given in parentheses.

4.4 Using SBI with other underlying branch predictors

Tables 6 and 7 show the PVN, SPEC and INVERSIONBENEFIT for SBI on top of McFarling and Bi-Mode branch predictors. Clearly, it is more difficult to apply SBI to these predictors because of lower PVN and SPEC. Although SBI improves the overall results, the improvement for these predictors is not as large as the improvement for the Gshare predictor.

Figures 12 and 13 show the misprediction rates from using SBI on top of McFarling and Bi-Mode branch predictors, respectively. The results were generated using the functional simulator. The predictor size includes the hardware required for SBI. Results show that with a similar hardware cost, we decrease mispredictions for both predictors by an average of 5%–7% for predictors using 10k bits or more. If hardware resources are limited, SBI with either predictor can be used to achieve similar misprediction rates with much less hardware. For example, the McFarling predictor with 48 kbits has a mean misprediction rate of 5.5%, while the predictor with SBI using 16k fewer bits produces a mean misprediction rate of 5.4%.

McFarling showed that a number of variations are possible for the combining predictor. One particular variation is used in the Alpha 21264 processor [6]. The 21264 uses a McFarling predictor with a 4k-entry meta predictor indexed by global history, a 4k-entry global predictor, also indexed by global history, and a SAg with a 1k-entry, 10-bit BHR and a 1k-entry PHT. The SAg [14]

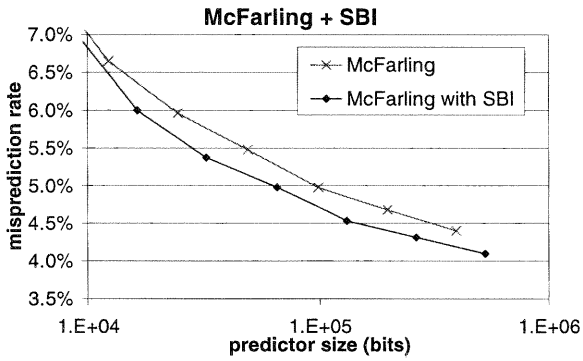


Figure 12: McFarling predictor misprediction rate (geometric mean) with and without SBI. Results shown for McFarling with SBI include the hardware used for the JRS confidence estimator.

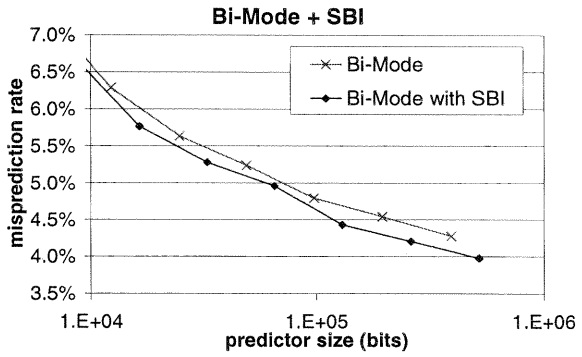


Figure 13: Bi-Mode predictor misprediction rate (geometric mean) with and without SBI. Results shown for Bi-Mode with SBI include the hardware used for the JRS confidence estimator.

uses 3-bits per PHT entry, while the global and meta predictors use 2-bits per PHT entry. The global history is updated speculatively, and the local history is updated non-speculatively. Table 8 shows the geometric mean of the misprediction rate for the 21264 configuration. The numbers were generated using the functional simulator. The table also shows the misprediction rate for a *modified* version, which uses a 2k-entry meta table and a 2k-entry global predictor. This decreases the hardware cost to 21 kbits and increases the misprediction rate from 5.47% to 5.91%. Finally, we implemented SBI on top of the *modified* 21264 configuration using a 4k-entry confidence estimator, resulting in a 29 kbit configuration. The SBI predictor out-performs the original 21264 configuration of equal hardware size. The mean misprediction rate with SBI is reduced to 4.97%, a relative improvement of 9% over the original 21264 predictor. These results show that even for highly optimized branch predictor configurations, SBI improves prediction accuracy with identical hardware cost.

4.4.1 Dynamic Inversion Monitoring

As mentioned above, both the PVN and SPEC numbers are lower for the Bi-Mode and McFarling predictors when compared to the Gshare predictor. For some Bi-Mode and McFarling sizes and SBI configurations, the average PVN numbers drop below 50%. For these situations, SBI results in a performance loss. Figure 14 shows the temporal behavior of prediction accuracy and PVN for the initial two million branches of the *Compress* benchmark using the Gshare predictor. Notice that PVN is almost always above 50%. By comparison, the time varying behavior of PVN for the McFarling predictor, shown in Figure 15, averages below 50%, with frequent excursions above 50%. For some programs, PVN may *always* be below 50%, while for others it may always be *above* 50%. Using a dynamic method to determine when SBI should be applied makes the SBI predictor more resilient to such program variations.

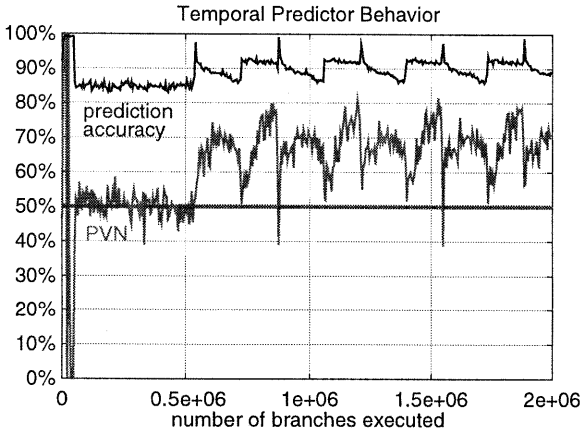


Figure 14: Time-varying values of PVN for Compress using the Gshare predictor.

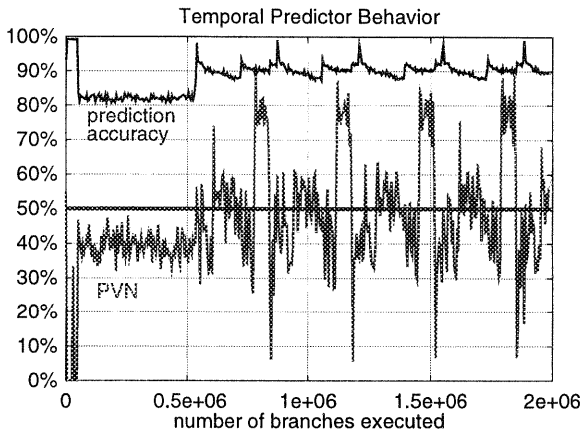


Figure 15: Time-varying values of PVN for Compress using the McFarling predictor.

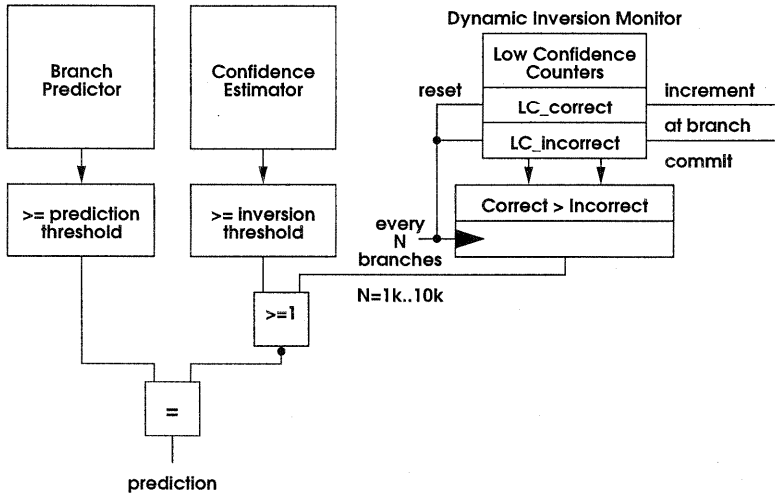


Figure 16: Schematic diagram showing how Dynamic Inversion Monitoring is combined with SBI.

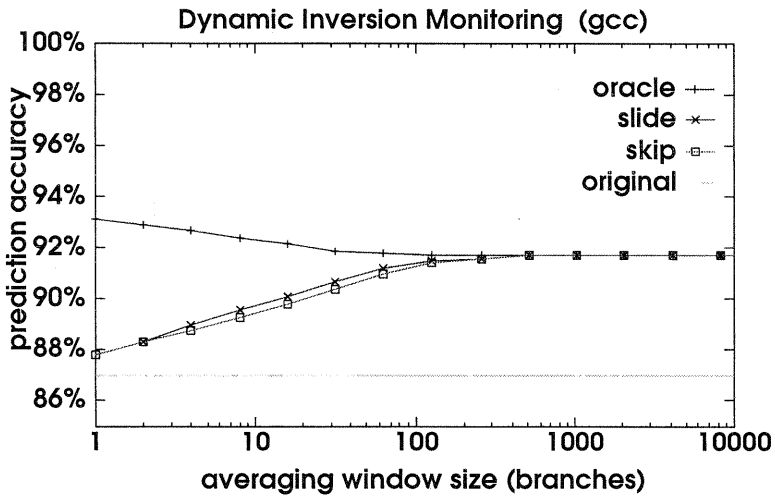


Figure 17: The effect of different window sizes on the dynamic SBI prediction accuracy.

We developed the Dynamic Inversion Monitoring (DIM) method to disable SBI when the PVN value drops below 50%. Figure 16 shows our modifications to the SBI predictor. As branches are committed, we count the number of correctly and incorrectly predicted low-confidence branches over a sampling period. If the number of incorrectly predicted, low-confidence branches was greater than the number of correctly predicted, low-confidence branches in the previous sampling period, (*i.e.* if the INVERSIONBENEFIT for the period was positive), then SBI is engaged; otherwise it is disengaged. However, the confidence estimator is always updated.

Figure 17 shows the effectiveness of four dynamic sampling organizations for varying sampling periods on the *Gcc* program. The line marked "original" is the original misprediction rate when SBI is not used. The line marked "oracle", uses the samples surrounding the current branch. This is impractical to implement because it requires knowledge about the future. The line marked "slide" uses a sliding window over past branch decisions; as each branch is committed, the counters are updated by removing the effect of the oldest branch and including the effects of the current branch. The line marked "skip" simply resets the counters at the end of a sampling period. We vary the sampling period length from 1 to 10k branches. Although oracle indicates that the smaller sampling periods would result in the best performance, the practical methods ("skip" and "slide") reach their maximum at a sampling period of $\approx$1,000 branches, and are as effective as the "oracle" mechanism at that window size. Smaller sampling periods result in too much statistical noise for practical methods. These results are consistent across all applications. We decided to use the "skip" mechanism with a 1,000-branch sampling period.

Table 9 shows the misprediction rates of a Bi-Mode branch predictor with and without DIM. All component predictors have 2k entries. The JRS uses 2-bit up/down counters with a threshold of 1 and is indexed with the branch PC and one history bit. The table shows that static SBI improves performance for 5 benchmarks, matches performance for 2 benchmarks, and loses compared to the baseline Bi-Mode predictor for *Perl*. With DIM, this performance loss is prevented and is turned into a slight improvement compared to the base case. The improvement results from the short times during the program execution when SBI was considered useful and was turned on successfully. This shows that DIM is effective in dynamically tracking SBI performance. DIM acts as a safeguard

	Misprediction Rate		
	Baseline (no SBI)	static SBI	Dynamic Inversion Monitoring
compress	8.69%	8.70%	8.65%
gcc	8.73%	7.94%	8.03%
perl	2.87%	3.20%	2.83%
go	22.38%	20.73%	20.82%
m88ksim	3.43%	3.30%	3.33%
lisp	5.71%	5.71%	5.70%
vortex	1.05%	0.91%	0.94%
jpeg	10.20%	9.93%	9.98%

Table 9: Performance effect of Dynamic Inversion Monitoring (DIM).

to prevent possible performance losses of SBI, e.g. in the case of *Perl*. Also note, that some of the other benchmarks show slight performance variations; DIM uses past SBI behavior to predict future SBI behavior, and different benchmarks show different amounts of temporal correlation.

5 Conclusions

In this paper, we have proposed the Selective Branch Inversion (SBI) scheme to improve misprediction rates for a variety of branch predictors. SBI incorporates a confidence estimator on top of an existing branch predictor. If the branch is low-confidence, i.e. the branch is believed to be mispredicted, then the branch prediction is inverted. Starting with the confidence estimator proposed in [4] and the evaluation method discussed in [2], we have developed a confidence estimator that is a viable mechanism for SBI. We have analyzed the SBI predictor on top of Gshare and have shown that SBI works because the confidence estimator can detect and correct mispredictions that are a result of destructive interference in the underlying branch predictor. Furthermore, we have shown that SBI can be improved with the aid of the Dynamic Inversion Monitor (DIM). DIM dynamically tracks the effectiveness of SBI and disables inversion when it is detrimental to overall prediction accuracy, which makes SBI resilient to program variations. We have incorporated SBI with Gshare, Bi-Mode and various McFarling predictors, and have demonstrated consistent improvements in prediction accuracy for similar or less hardware cost.

Acknowledgements

We would like to thank Digital Equipment Corporation for an equipment grant that provide the simulation cycles, support from Hewlett-Packard. This work was partially supported by NSF grants No. CCR-9401689, No. MIP-9706286 and in part by ARPA contract ARMY DABT63-94-C-0029.

References

[1] Doug Burger, Todd M. Austin, and Steve Bennett. Evaluating Future Microprocessors: The SimpleScalar Tool Set. Technical Report TR#1308, University of Wisconsin, July 1996.

[2] Dirk Grunwald, Artur Klauser, Srilatha Manne, and Andrew Pleszkun. Confidence Esimation for Speculation Control. In *25th Intl. Symp. on Computer Architecture*, Barcelona, Spain, June 1998. ACM, IEEE.

[3] Eric Hao, Po-Yung Chang, and Yale N. Patt. The Effect of Speculatively Updating Branch History on Branch Prediction Accuracy, Revisited. In *27th Annual Intl. Symp. on Microarchitecture*, San Jose, CA, December 1994. ACM, IEEE.

[4] Erik Jacobsen, Eric Rotenberg, and J. E. Smith. Assigning Confidence to Conditional Branch Predictions. In *29th Annual Intl. Symp. on Microarchitecture*, pages 142–152, Paris, France, December 1996. ACM, IEEE.

[5] Toni Juan, Sanji Sanjeevan, and Juan J. Navarro. Dynamic History-Length Fitting: A Third Level of Adaptivity for Branch Prediction. In *25th Intl. Symp. on Computer Architecture*, Barcelona, Spain, June 1998. ACM, IEEE.

[6] R. E. Kessler, E. J. McLellan, and D. A. Webb. The Alpha 21264 Microprocessor Architecture. In *International Conference on Computer Design*, October 1998.

[7] Artur Klauser, Abhijit Paithankar, and Dirk Grunwald. Selective Eager Execution on the PolyPath Architecture. In *25th Intl. Symp. on Computer Architecture*, Barcelona, Spain, June 1998. ACM, IEEE.

[8] Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge. The Bi-Mode Branch Predictor. In *30th Annual Intl. Symp. on Microarchitecture*, Research Triangle Park, NC, December 1997. ACM, IEEE.

[9] Srilatha Manne, Artur Klauser, and Dirk Grunwald. Pipeline Gating: Speculation Control for Energy Reduction. In *25th Intl. Symp. on Computer Architecture*, Barcelona, Spain, June 1998. ACM, IEEE.

[10] Scott McFarling. Combining Branch Predictors. WRL TN-36, Digital Western Research Lab, June 1993.

[11] Pierre Michaud, André Seznec, and Richard Uhlig. Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. In *24th Intl. Symp. on Computer Architecture*, Denver, CO, June 1997. ACM, IEEE.

[12] Stuart Sechrest, Chih-Chieh Lee, and Trevor Mudge. Correlation and Aliasing in Dynamic Branch Predictors. In *23rd Intl. Symp. on Computer Architecture*, pages 22–32. ACM, IEEE, June 1996.

[13] Eric Sprangle, Robert S Chappell, Mitch Alsup, and Yale N. Patt. The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference. In *24th Intl. Symp. on Computer Architecture*, Denver, CO, June 1997. ACM, IEEE.

[14] Tse-Yu Yeh and Yale N. Patt. Alternative Implementations of Two-Level Adaptive Branch Predictions. In *19th Intl. Symp. on Computer Architecture*, pages 124–134. ACM, IEEE, June 1992.

[15] Cliff Young, Nicholas Gloy, and Michael D. Smith. A comparative analysis of schemes for correlated branch prediction. In *22nd Intl. Symp. on Computer Architecture*, pages 276–286, Santa Margherita, Italy, June 1995. ACM, IEEE.